

Configurando o ambiente de testes unitários em aplicações de micro frontend no NX

ENTENDO A ESTRUTURA DO PROJETO

ÁRVORE DO PROJETO (ARQUIVOS)

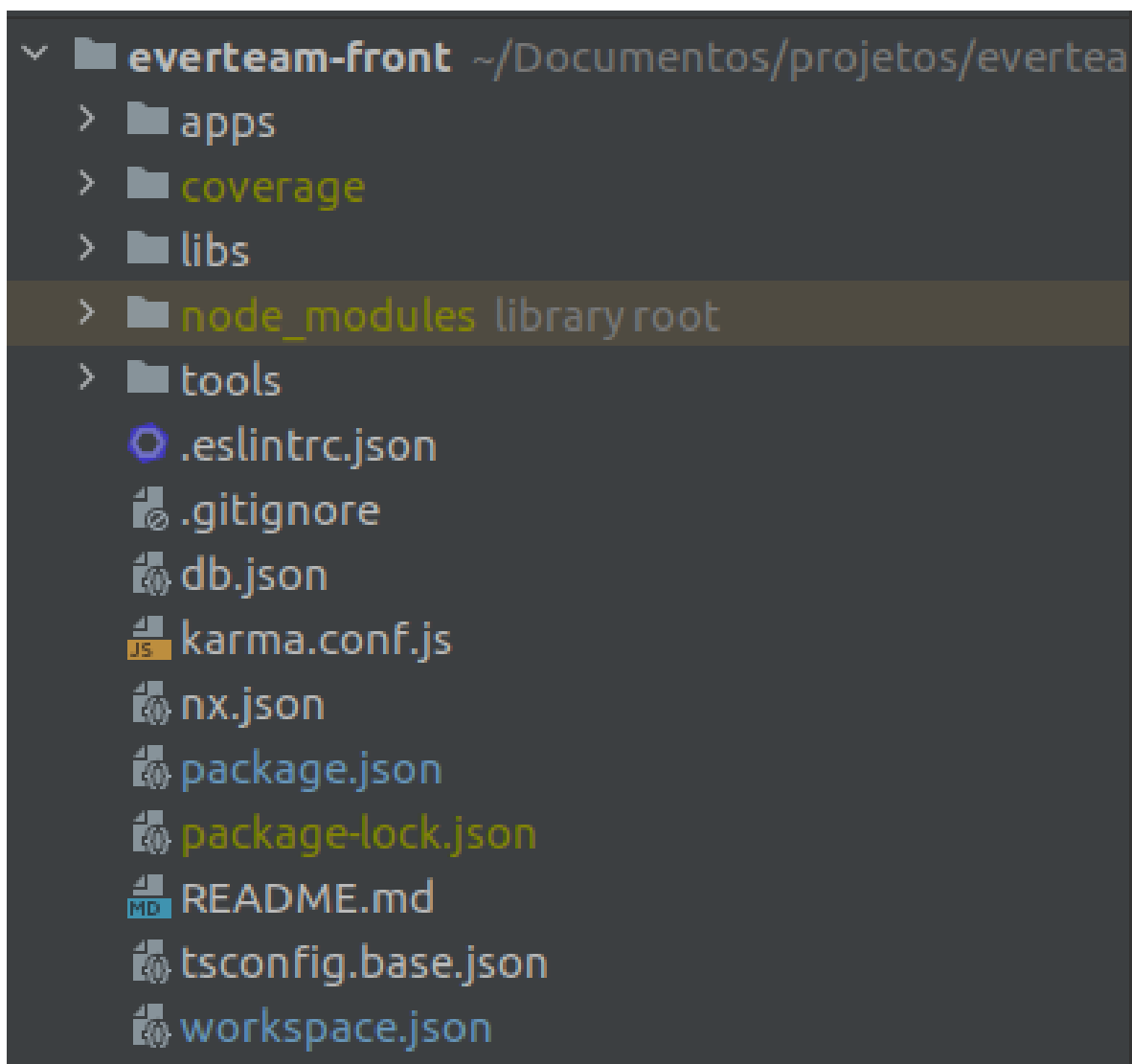


Figura 1 - Árvore do projeto

NOTA

Independente do projeto de micro front-end que você esteja atuando, ele seguirá o modelo apresentado acima ou com poucas diferenças.

A grosso modo temos que o **workspace.json** é similar ao nosso **angular.json** numa aplicação front-end normal.

ENTENDO A ESTRUTURA DO PROJETO

PACKAGE.JSON

Para termos uma estrutura e os testes sejam executados de forma correta em nossa aplicação devemos ter alguns pacotes instalados.

Dica: copie o comando **npm** a seguir para ficar mais fácil.

```
npm install karma karma-chrome-launcher karma-coverage karma-jasmine karma-jasmine-html-reporter karma-spec-reporter @types/jasmine @types/jasminewd2 -D
```

Com os pacotes instalados, na raiz do projeto como pode ver na figura 1, crie o arquivo **karma.conf.js**

NOTA

Se este arquivo estiver criado apenas confira as informações conforme o código mais abaixo.

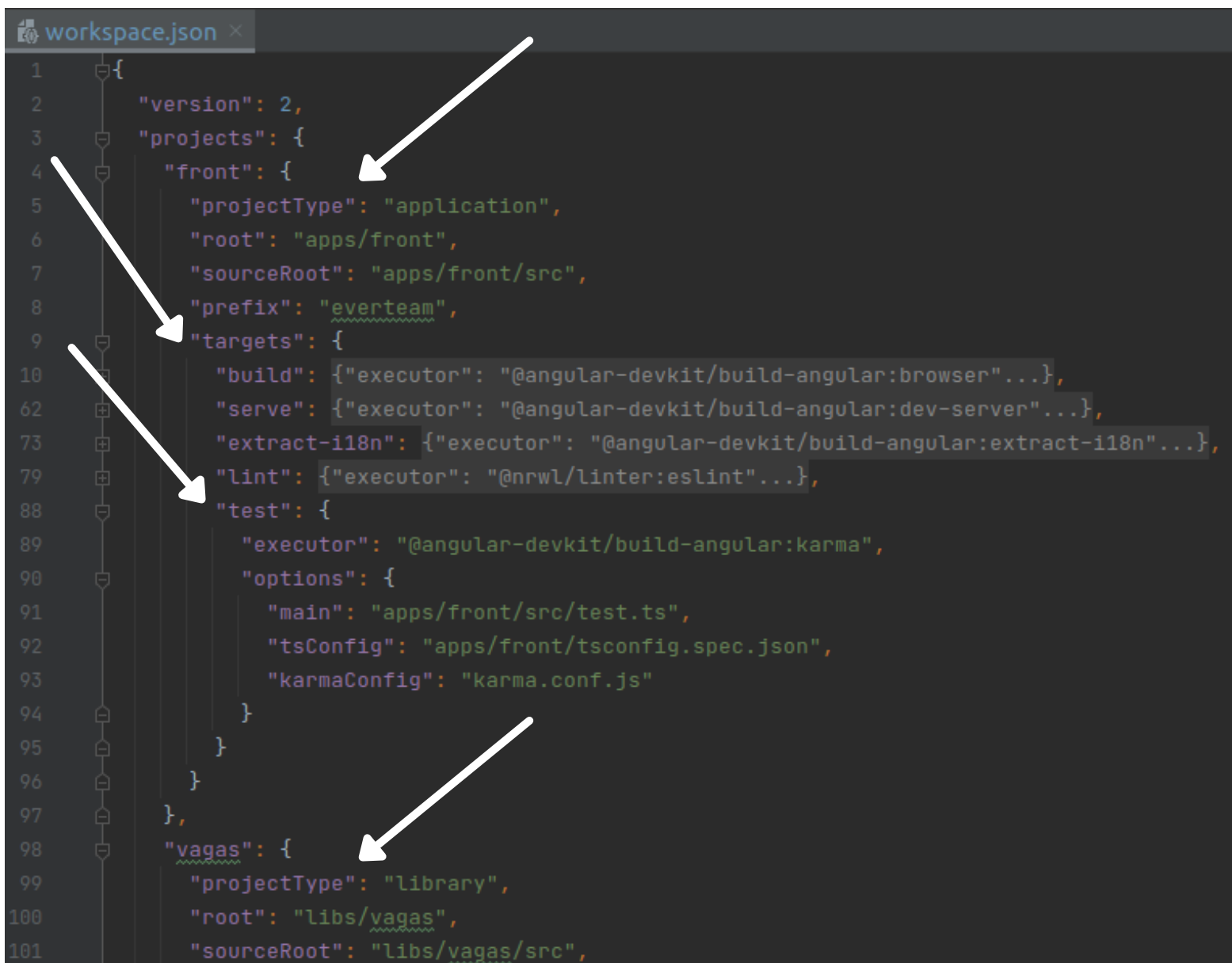
Caso não esteja criado, faça sua criação e insira as informações do código mais abaixo.

ENTENDO A ESTRUTURA DO PROJETO

KARMA.CONF.JSON

```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage'),
      require('karma-spec-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    files: [],
    exclude: [],
    preprocessors: {},
    reporters: ['spec'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['ChromeHeadless'],
    singleRun: false,
    concurrency: Infinity,
    restartOnFileChange: true,
    client: {
      cliContext: true
    },
    coverageReporter: {
      type: 'html',
      dir: 'coverage/',
      subdir: '.'
    }
  });
}
```

ENTENDENDO O WORKSPACE.JSON

A screenshot of a code editor showing the contents of a file named 'workspace.json'. The file is a JSON object with a 'version' of 2 and a 'projects' array. The first project is 'front', which is an 'application' with a 'root' of 'apps/front', a 'sourceRoot' of 'apps/front/src', a 'prefix' of 'everteam', and a 'targets' object. The 'targets' object contains 'build', 'serve', 'extract-i18n', 'lint', and 'test' tasks, each with an 'executor' and 'options'. The second project is 'vagas', which is a 'library' with a 'root' of 'libs/vagas' and a 'sourceRoot' of 'libs/vagas/src'. Four white arrows point to specific parts of the JSON: one to the 'version' field, one to the 'front' project object, one to the 'targets' object, and one to the 'vagas' project object.

```
1  {
2    "version": 2,
3    "projects": {
4      "front": {
5        "projectType": "application",
6        "root": "apps/front",
7        "sourceRoot": "apps/front/src",
8        "prefix": "everteam",
9        "targets": {
10         "build": {"executor": "@angular-devkit/build-angular:browser"...},
11         "serve": {"executor": "@angular-devkit/build-angular:dev-server"...},
12         "extract-i18n": {"executor": "@angular-devkit/build-angular:extract-i18n"...},
13         "lint": {"executor": "@nrwl/linter:eslint"...},
14         "test": {
15           "executor": "@angular-devkit/build-angular:karma",
16           "options": {
17             "main": "apps/front/src/test.ts",
18             "tsConfig": "apps/front/tsconfig.spec.json",
19             "karmaConfig": "karma.conf.js"
20           }
21         }
22       }
23     },
24     "vagas": {
25       "projectType": "library",
26       "root": "libs/vagas",
27       "sourceRoot": "libs/vagas/src",
```

Figura 2 - workspace.json

No **workspace.json** ele é dividido em projetos e seu tipo deve ser evidenciado como sendo **application** ou **library**.

Neste exemplo temos **front(application)** e **vagas(library)**.

No **targets de front** temos: build, serve, extract-i18n, lint e test. Estes **targets** são usados pelo compilador para ler as instruções de execução e quais arquivos buscar para esta tarefa.

ENTENDENDO O WORKSPACE.JSON

Devemos nos atentar de **test** pois é nele que devemos informar qual pacote npm vai ser executado e passar as opções que são primordiais:

- **main:** arquivo **test.ts** que precisa estar dentro do **src** seja do **application** ou **library**.
- **tsConfig:** arquivo **tsconfig.spec.json** que configura qual biblioteca de execução das assertividades, tipos de arquivos aceitos e a saída para a convergência, etc.
- **karmaConfig:** é o arquivo na raiz de todo o projeto que devemos deixar explícito.

Veja abaixo como fica o exemplo da árvore da aplicação front com esses arquivos.

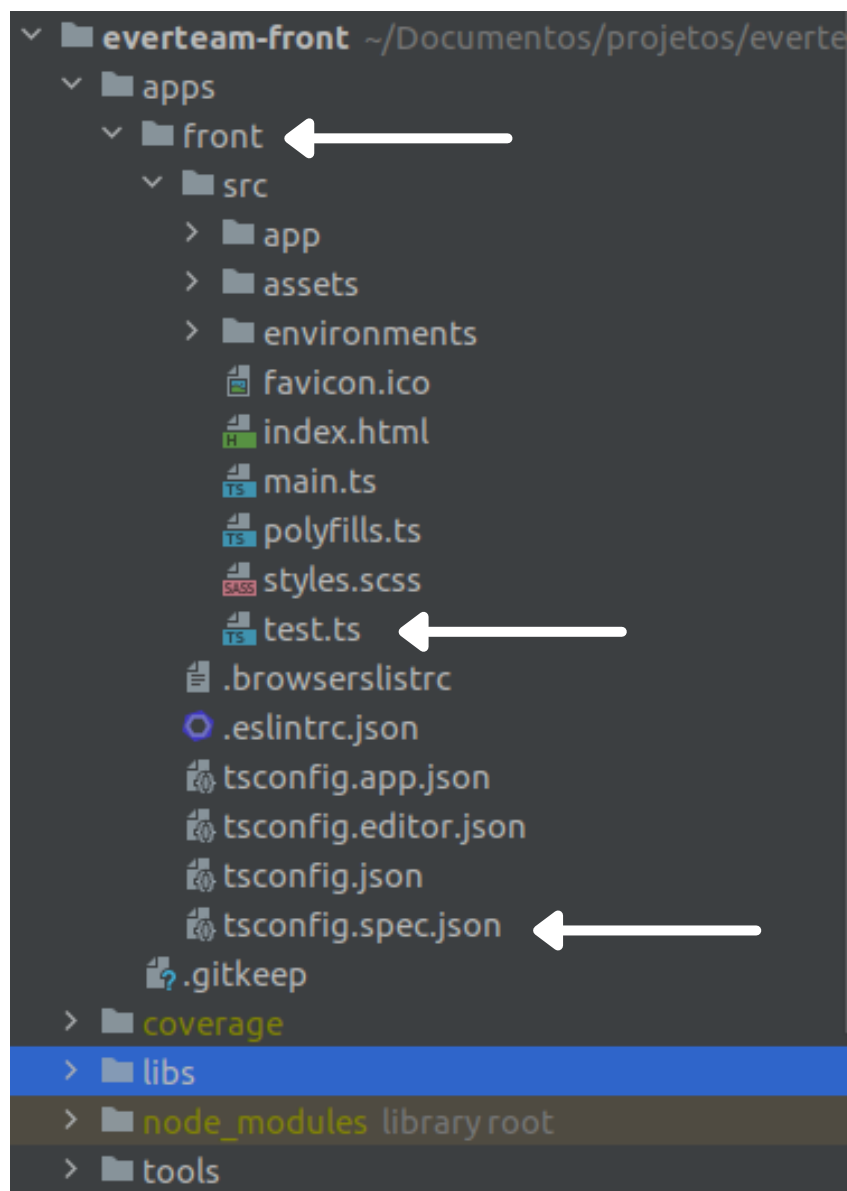


Figura 3 - árvore de arquivo do front

ENTENDENDO O WORKSPACE.JSON

Veja abaixo como fica o exemplo da árvore da **lib vagas** com esses arquivos.

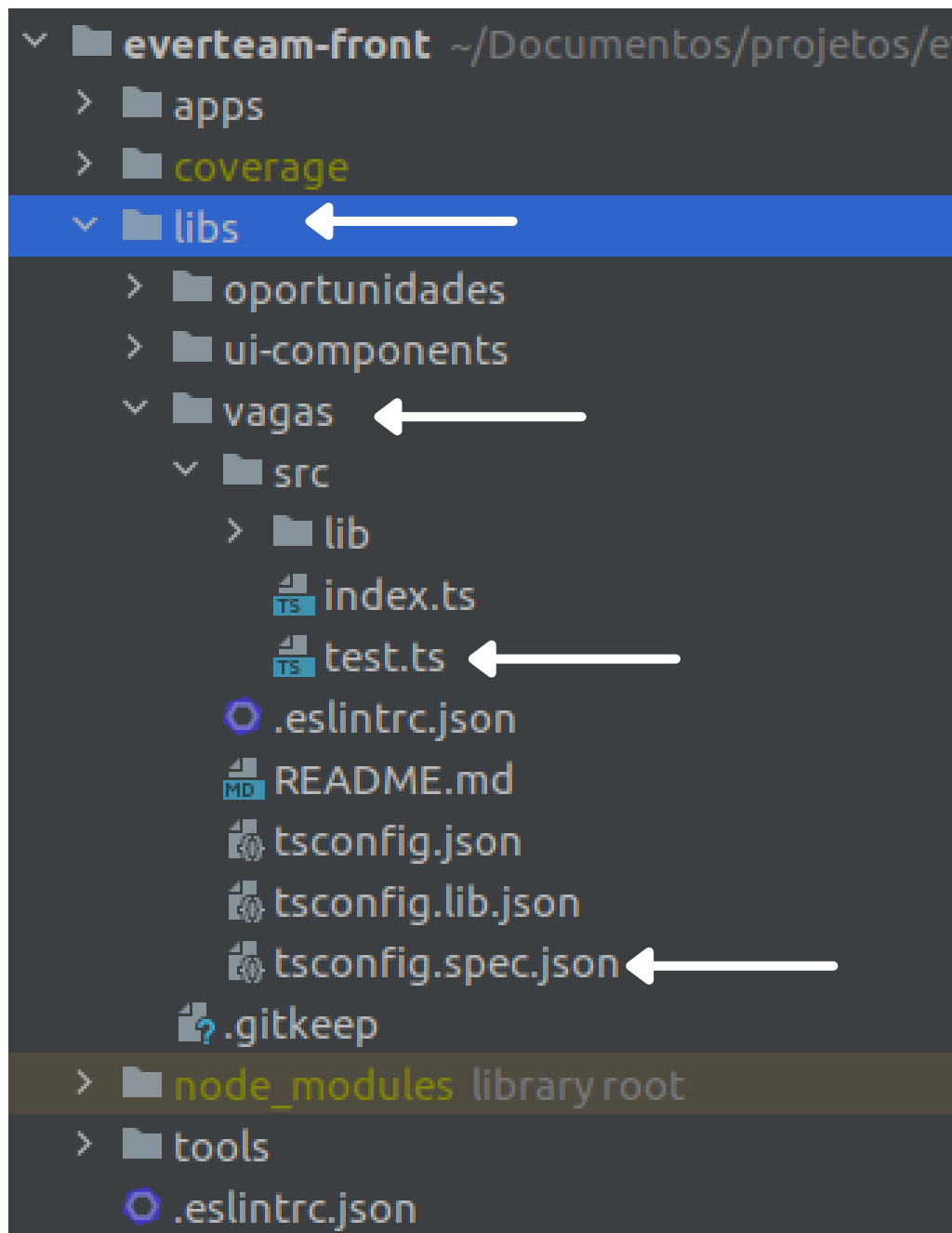


Figura 4 - árvore de arquivo da lib vagas

No **workspace.json** o target ficará como na figura 5.

Lembrando que para cada nova lib criada deverá ser inserido o target de test para a mesma, senão o teste quando for executado não vai iniciar.

ENTENDENDO O WORKSPACE.JSON

```
98     "vagas": {
99         "projectType": "library",
100         "root": "libs/vagas",
101         "sourceRoot": "libs/vagas/src",
102         "prefix": "everteam",
103         "targets": {
104             "lint": {
105                 "executor": "@nrwl/linter:eslint",
106                 "options": {
107                     "lintFilePatterns": [
108                         "libs/vagas/src/**/*.ts",
109                         "libs/vagas/src/**/*.html"
110                     ]
111                 }
112             },
113             "test": {
114                 "executor": "@angular-devkit/build-angular:karma",
115                 "options": {
116                     "main": "libs/vagas/src/test.ts",
117                     "tsConfig": "libs/vagas/tsconfig.spec.json",
118                     "karmaConfig": "karma.conf.js"
119                 }
120             }
121         }
122     },
```

Figura 5 - target de test para a lib vagas

O conteúdo dos arquivos **test.ts** e **tsconfig.spec.json** será como dos códigos abaixo.

NOTA

Para as **libs** pode usar o mesmo, já que é um arquivo de configuração.

ARQUIVOS QUE ESTÃO PRESENTE NAS LIBS

TEST.TS

```
import 'zone.js/dist/zone';
import 'zone.js/dist/zone-testing'

import { getTestBed } from '@angular/core/testing';
import { BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

declare const require: any;

getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);

const context = require.context('./', true, /\.spec\.ts$/);
context.keys().map(context);
```

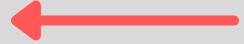
TSCONFIG.SPEC.JSON

```
{
  "extends": "../../tsconfig.base.json",
  "compilerOptions": {
    "outDir": "../../dist/out-tsc",
    "types": ["jasmine", "node"]
  },
  "files": ["src/test.ts"],
  "include": ["**/*.spec.ts", "**/*.d.ts"]
}
```

TSCONFIG.LIB.JSON

Este arquivo estará presente dentro de cada lib e nele inclua o seguinte item na propriedade **"include"**.

```
"include": [  
  "**/*.ts",  
  "../../node_modules/@types/jasmine/index.d.ts"  
]
```



Isto servirá para que o lint não fique dando warning para não reconhecer os types do jasmine como por exemplo: describe, beforeEach, it.

EXECUÇÃO DOS TESTES

Para executar os testes você tem algumas opções:

1. Ter um script no package.json para cada lib criada:

```
5   "scripts": {  
6     "nx": "nx",  
7     "start": "nx serve",  
8     "build": "nx build",  
9     "test": "nx test",  
10    "test:vagas": "nx test vagas --watch=false --codeCoverage=true",
```

Para executar o comando no terminal ou git bash basta digitar:

```
npm run test:vagas
```

--watch=false, serve para não visualizar os logs gerados.
--codeCoverage=true, serve para convergir os testes e deixar disponível na **pasta de coverage**.

2. Executar diretamente no terminal ou git bash:

```
nx test vagas
```

NOTA

Fique a vontade para inserir as flags abaixo quando executar pelo terminal ou git bash:

```
--watch=false
--codeCoverage=true
```

EXEMPLE

```
nx test vagas --watch=false --codeCoverage=true
```

IMPLEMENTAÇÃO DOS TESTES

Talvez para você seja novidade a ideia de escrever testes unitários. Para isto eu desenvolvi um artigo que está no LinkedIn que ajuda nessa introdução ao mundo de testes.

A escrita de testes em ambiente de micro front-end é a mesma quando em um front-end comum, então os conceitos são reaproveitados.

Não se esqueça de curtir o artigo se ele lhe ajudar de alguma forma.

Link, copie e cole no navegador:

<https://www.linkedin.com/pulse/testes-unit%C3%A1rios-de-service-e-component-em-angular-firmino-lemos/>